A flow-first route-next heuristic for liner shipping network design

Krogsgaard, Alexander; Pisinger, David; Thorsen, Jesper

Published in:
Networks

Link to article, DOI:
10.1002/net.21819

Publication date:
2018

Document Version
Peer reviewed version

A flow-first route-next Heuristic for Liner Shipping Network Design

Alexander Krogsgaard, David Pisinger, Jesper Thorsen

Abstract Having a well-designed liner shipping network is paramount to ensure competitive freight rates, adequate capacity on trade-lanes, and reasonable transportation times. The most successful algorithms for liner shipping network design make use of a two-phase approach, where they first design the routes of the vessels, and then flow the containers through the network in order to calculate how many of the customers demands can be satisfied, and what the imposed operational costs are. In this paper we reverse the approach by first flowing the containers through a relaxed network, and then design routes to match this flow. This gives a better initial solution than starting from scratch, and the relaxed network reflects the ideas behind a physical internet of having a distributed multi-segment intermodal transport. Next, the initial solution is improved by use of a variable neighborhood search method, where six different operators are used to modify the network. Since each iteration of the local search method involves solving a very complex multi-commodity flow problem to route the containers through the network, the flow problem is solved heuristically by use of a fast Lagrange heuristic. Although the Lagrange heuristic for flowing containers is 2–5% from the optimal solution, the solution quality is sufficiently good to guide the variable neighborhood search method in designing the network. Computational results are reported, showing that the developed heuristic is able to find improved solutions for the large-scale world instances from LINERLIB, and it is the first heuristic to report results for the biggest WorldLarge instance.

Keywords Network Design, Liner Shipping, Multi-commodity Flow, Local Search, Lagrange heuristic

DTU Management Engineering, Technical University of Denmark, Produktionstorvet 424, DK-2800 Kgs. Lyngby, Denmark

1 Introduction

The liner shipping industry is a vital part of the global economy, facilitating much international trade by transporting large amounts of cargo around the world at low cost. During the last three decades the volume of containerized cargo has grown more than 8% per year (Unctad, 2015).

Even though the environmental impact of container transport per ton-km is very low compared to other transport modes, the industry still contributes significantly to the global CO2 emissions due to the large volumes transported. In 2012, the international shipping industry was estimated to account for 2.2% of the global CO2 emissions, of which approximately a quarter was caused by container vessels (IMO, 2014). In 2015, there were around 5,100 container vessels in operation worldwide according to Unctad (2016).

In this paper we present a heuristic for solving the Liner Shipping Network Design Problem (LSNDP) from scratch — without an already existing network as a starting point. The objective is to maximize the profit composed of revenue minus operational costs. A liner shipping network consists of a number of rotations, which is a scheduled roundtrip between a set of ports. Rotations are preferably operated weekly, even though examples of biweekly rotations can also be found. A rotation is operated by a number of vessels to achieve the desired frequency.

In larger networks with many ports, it is in practice impossible to connect all ports with direct connections. Some containers will thus have to be transshipped to get from origin to destination, meaning that the container is switched from one rotation to another at an intermediate port. A transshipment is associated with an extra cost for handling the container.

The capacity of a vessel can be measured by the number of containers, it can carry, which can be calculated in TEU or FFE. One TEU is a 20-foot container equivalent, while one FFE is a 40-foot container equivalent, hence two TEU equals one FFE. Container vessels exist in many different sizes, from less than 500 FFE for the smallest feeder vessels up to almost 10,000 FFE for the largest vessels. Due to economies of scale, the cost of transporting a container is lowest on large vessels, but smaller vessels can enter more ports and canals.

To be able to operate a shipping network, it must be decided how to flow the containers — which containers should go on which vessels, where will transshipments occur etc. The containers share the capacity of the vessels, but have unique origins and destinations, and it is thus necessary to solve a multicommodity flow problem (MCF). Apart from being a prerequisite for operating the network, the MCF also determines a significant part of the cost and all of the revenue, in that it can be decided to reject cargo. The MCF thus has to be solved when evaluating the network, and the quality of the solution to the MCF can easily make the difference between a profitable and an unprofitable network. Because of this, the MCF is a vital subproblem to the LSNDP.

1.1 Previous work

For a detailed review of the research conducted within liner shipping optimization problems, the survey papers Ronen (1983), Ronen (1993), Christiansen et al. (2004), Christiansen et al. (2013), Meng et al. (2014), Brouer et al. (2016, 2017), and Lee and Song (2017) can be recommended.

Agarwal and Ergun (2008) present three different methods for designing liner ship networks with up to 100 vessels and 20 ports. Heuristic, column generation and Benders decomposition methods are applied with the most encouraging results obtained by column generation and Benders decomposition. Apart from the rather small instance size compared to real world networks, the main drawback of the presented model is that transshipment costs are not taken into account. As transshipments are a vital part of the liner shipping business, this simplification is unfortunate.

Transshipment costs are included in the work by Alvarez (2009). As ports are represented by a single node, however, the model cannot detect transshipments occurring within the same rotation if a rotation visits the same port multiple times, which reduces the accuracy of the cost calculation. In addition to this, it is assumed that the cost of transshipping a container is equal to the cost of loading and unloading a container, which might often not be the case. It is, for example, easier for an operator to switch transshipment port than to change the origin or destination of cargo, making transshipment costs more competition prone than load and unload costs.

In contrast to Agarwal and Ergun, Alvarez imposes no frequency restriction on the rotations, but allows any integer number of vessels to be assigned to each rotation. This means that a rotation can be, for example, 20 days long, and will be operated every 20 days if one vessels is assigned, every 10 days if two vessels are assigned etc. As described by Brouer et al. (2014a), business rules of operators make this assumption quite unrealistic, as a rotation must meet a basic requirement of either weekly or biweekly frequency, which also fixes rotation lengths at whole weeks. If this requirement is not met, the schedule will be unpredictable for shippers with changing days of service and low frequency, and thus commercially unattractive. The frequency requirement is very important for the generation of rotations, as it prevents a solution with many rotations operating infrequently, but together providing a high number of direct connections between ports and thus saving transshipment costs.

Alvarez uses a two-stage approach for solving the problem. In each iteration, the multicommodity flow problem (MCF) is solved to determine the cost of the current network. Based on the resulting flow, the network is optimized using a tabu search algorithm. The MCF is solved to optimality, which is reported to be time-consuming for an instance of only seven ports.

Brouer et al. (2014a) presents a base integer programming model for the LSNDP. This model includes the requirement of at least biweekly frequency and also models transshipment costs more accurately than Alvarez by keeping this cost separate from the load and unload costs. The rotations are restricted to having at most one port

called twice, while all other ports can be called only once, which is a restriction not included by Alvarez. Although allowing one port to be visited twice, resulting in a so-called butterfly rotation, is better than prohibiting multiple calls at the same port altogether, it is still a restriction not found in real world rotations.

Brouer et al. (2014a) solves the problem in two stages like Alvarez. In each iteration, the MCF is solved and new rotations are generated based on the resulting flow. The rotations are collected in a candidate list from which the most promising are selected to make up the network. This is controlled by a tabu search algorithm. Brouer et al. (2014a) also provides data on instances through LINER-LIB, which is publicly available. This data includes distances between ports, demand matrices and vessel information for seven different instances, which can be used for benchmarking results. Brouer et al. (2014a) is able to solve six of seven instances, while the largest instance remains unsolved. For the instances solved, running times vary from five minutes for the smallest instance to four hours for the largest.

In Brouer et al. (2014b) the two-stage framework is developed further. Since it is so time-consuming to solve the MCF problem, larger moves are made in each iteration. These moves are identified by solving a smaller mathematical model, using a number of estimates for predicting the consequences of inserting or omitting a port. The resulting *mathheuristic* reports improved solution quality compared to Brouer et al. (2014a).

Recently, Karsten et al. (2016) included time constraints in the LSNDP. This extension provides a more customer-oriented and commercial approach to the LSNDP in the realization that demand is not independent of transit times. If the offered product is too slow, shippers will choose another operator. Karsten et al. (2016) includes a preset time limit for each OD pair such that the demand goes to zero if the transit time limit is exceeded. Although it can be argued that such a hard time limit is too simplistic, it is nevertheless an improvement over the LSNDP without time constraints, where any transit time is acceptable.

1.2 Solution approach and contribution

As previously discussed, the most successful algorithms for liner shipping network design make use of a two-phase approach, where they first design the routes of the vessels, and then solve a MCF to flow the containers through the network and calculate operational costs.

In this paper we reverse the approach by first flowing the containers through a relaxed network, and then design rotations to match this flow. To capture the hub structure, which is vital in a liner shipping network, the rotations will be generated based on an initial flow, where cargo is flown through a complete network with all connections between ports available. The connections are priced such that they are expensive at low loads and cheap at high loads, in order to make the cargo gather at fewer connections. Sun and Zheng (2016) used a similar approach for finding hub locations in liner shipping. The initial flow can be seen as an accomplishment of the

physical internet where point-to-point transport has been replaced by multi-segment intermodal transport (Montreuil, 2011).

After having constructed an initial network, an improvement algorithm based on variable neighborhood search is used. Six different operators are used to modify the network. In order to quickly evaluate moves, a subgraph containing only the rotation being modified is introduced, enabling a fast solution to the MCF for this single rotation.

Since each iteration of the local search method involves solving a very complex multi-commodity flow problem to route the containers through the network, the flow problem is solved heuristically by use of a fast Lagrange heuristic. Although the Lagrange heuristic for the MCF is 2–5% from the optimal solution, the solution quality is sufficiently good to guide the variable neighborhood search method in designing the network. To obtain a usable flow, a repair-flow algorithm is devised to supplement the Lagrange heuristic. The general idea of the repair-flow algorithm is to gradually turn a flow, where the capacity constraints on the main edges is violated, into a valid flow through a heuristic method.

As opposed to previous algorithms both the MCF and the route construction is solved by fast heuristics. This makes it possible to search considerably more candidate networks within a given time frame, hence leading to a much more diversified search. Apart from more iterations, a shorter solution time to the MCF will also allow a more comprehensive graph structure, which can both represent transshipments accurately and allow complex rotations without restrictions on the number of times a port is visited. It will thus be possible to generate more realistic rotations, bringing the solutions closer to those seen in real life.

1.3 Test instances

We will test the developed algorithms on the LINER-LIB instances presented by Brouer et al. (2014a). An overview of the instances is given in Table 1.

Instance	Instance type	Ports	Demands	Vessels	Total weekly demand (FFE)	Total vessel capacity (FFE)
Baltic	Single-hub	12	22	6	4904	3400
WestAfrica (WAF)	Single-hub	20	37	42	8541	28700
Mediterranean	Multi-hub	40	365	20	7545	14800
Pacific	Trade lane	45	722	100	44180	151800
AsiaEurope	Trade lane	114	4000	176	76944	425900
WorldSmall	Multi-hub	47	1764	263	136387	611800
WorldLarge	Multi-hub	202	9615	501	138914	1071100

Table 1 The seven different instances included in LINER-LIB

In general, the small instances Baltic, WestAfrica, Mediterranean and Pacific will be used for tuning the algorithms, while the larger instances AsiaEurope, WorldS-

mall and WorldLarge will be used for the final test of the algorithm. In some cases we will, however, also use WorldSmall for tuning in order to ensure a proper functionality for large instances.

All instances exist in a *base*, *high* and *low capacity* variant. The base variant will be used in this paper. All experiments have been run on a Intel Core i7-3770 CPU, 3.40GHz processor with 16GB RAM.

1.4 Outline

Section 2 describes the graph representation and defines the problem formally. Section 3 presents the solution method for flowing containers through a network. The algorithm is based on solving a MCF through a Lagrange heuristic. Next, Section 4 explains how a flow-first approach is used to construct an initial backbone flow, that is used for generating the initial rotations. Section 5 describes how the rotations are optimized by using a variable neighborhood search method. A subgraph to each rotation is used to evaluate the effect of inserting and removing ports. This is followed by Section 6 where the developed algorithms are tested on instances from LINERLIB, and compared to previous results. Section 7 concludes the paper and discuss perspectives on future work.

2 Problem definition

The objective of the LSNDP is to design a liner shipping network that maximizes profit. Brouer et al. (2014a) presents a mathematical model for the problem, which is based on three-index decision variables recording both the preceding and next port of each commodity. This is necessary to allow accurate representation of transshipments when each port is represented by a single node only. The formulation limits each rotation to a single butterfly port, i.e. a port included twice, while all other ports can be included only once. To allow more flexible generation of rotations with multiple duplicate ports, the graph representation used here contains several nodes for each port. Each port is represented by at least two nodes — one centroid node for departing commodities and one for arriving commodities. For every call at the port, two more nodes are added, one representing the arrival and one representing the departure. By using this representation, transshipments can be handled correctly without extra indices on the decision variables, but this requires an alteration of the mathematical model presented by Brouer et al. (2014a). The sets, parameters and decision variables of this altered model are defined in Table 2.

The rotations can be described by a directed graph $G = (N, A)$ with N being the set of nodes and A the set of arcs. A port in the LSNDP network is represented by two different types of nodes:

Sets	
V	Set of available vessel classes
R	Set of feasible rotations
N	Set of all nodes of the graph
N^r_{arr}	Set of arrival nodes for rotation $r \in R$
N^r_{dep}	Set of departure nodes for rotation $r \in R$
A	Set of all available arcs of the graph
A_r	Arcs representing rotation $r \in R$
K	Commodities to be transported between a pair of ports
Parameters	
d_k	Demand for commodity $k \in K$
r_k	Rate for commodity $k \in K$
O_k	Origin node of commodity $k \in K$
D_k	Destination node of commodity $k \in K$
u_v	Capacity of vessel class $v \in V$
z_v	Available number of vessels for vessel class $v \in V$
v_r	Vessel class used in rotation $r \in R$
n_r	Number of vessels required for operating rotation $r \in R$
$\overline{c}_r$	Fixed cost of operating rotation $r \in R$
c_{ij}	Variable cost of shipping one FFE on arc $(i,j) \in A$
Decision variables	
x^k_{ij}	The number of commodity $k \in K$ to be transported on arc $(i,j) \in A$
y_r	(binary) 1 iff rotation $r \in R$ is used in the solution

Table 2 Notation used in the LSNDP model

- *Centroids*: For each port $p \in P$ we have a source node (origin) and a sink node (destination) to handle the demands. The sources only have outgoing arcs, while the sinks only have ingoing arcs. The set of source nodes is S_α and the set of sink nodes is S_β.
- *Port calls*: For each vessel calling a port we introduce an arrival and departure node. Port call nodes are defined for each call q in the port call list Q^r for each rotation $r \in R$. The set of arrival nodes for rotation r is N^r_{arr} while the set of departure nodes is N^r_{dep}.

The set of arcs $A = A_s \cup A_t \cup A_l \cup A_u \cup A_d \cup A_o$ consists of the following subsets:

A_s (*sail arcs*) These arcs represent a rotation sailing from a departure node of one port to an arrival node of another port. The set of these arcs is defined as $A_s = \{(i,j) : i \in N^r_{dep}, j \in N^r_{arr}, r \in R\}$, where i and j are consecutive nodes, corresponding to consecutive calls in the port call list Q^r for the rotation $r \in R$. The cost associated with these arcs is zero. This is due to the assumption that vessels are deployed to support the rotation regardless of how many containers are transported. The capacity of these arcs is equal to that of the vessel class used in the given rotation, denoted u_{v_r}.

A_t (*transshipment arcs*) These arcs are used to represent the transshipment between two rotations in a port. They are connected from the arrival node of one rotation to the departure node of another rotation. The set of transshipment arcs is defined as

$A_t = \{(i,j) : i \in N^{rp}_{arr}, j \in N^{zp}_{dep}, p \in P, r,z \in R, r \neq z\}$. The cost of a transshipment arc corresponds to the transshipment cost at the given port denoted c^p_t.

A_l, A_u (*load/unload arcs*) These arcs are used from and to the centroids. Sources are connected to departure nodes, while arrival nodes are connected with sinks, within the same ports. Load arcs are defined as the set $A_l = \{(\alpha^p, i) : i \in N^p_{dep}, p \in P\}$, while unloading arcs are defined as the set $A_u = \{(i, \beta^p) : i \in N^p_{arr}, p \in P\}$. Here α^p and β^p are the source and sink nodes, respectively, while N^p_{arr} and N^p_{dep} are the arrival and departure nodes, respectively, for port $p \in P$. The cost for loading or unloading is given for each port as input data, and is denoted c^p_m.

A_d (*dwell arcs*) These arcs connect the arrival and departure node for a port call q. The set of dwell arcs is defined as $A_d = \{(i,j) : i \in N^q_{arr}, j \in N^q_{dep}, q \in Q^r, r \in R\}$, where N^q_{dep} and N^q_{arr} refer to the departure and arrival node for port call q. The cost of these arcs is zero, as a container that stays aboard the ship is assumed not to need handling. The capacity is equivalent to that of the vessel class used in the given rotation.

A_o (*omission arcs*) Since we cannot be sure that all containers can be shipped through the network, omission arcs connect all sources with all sinks with infinite capacity. Omission arcs are defined as the set $A_o = \{(i,j) : i \in S_\alpha, j \in S_\beta\}$, where S_α and S_β are the previously mentioned sets for sources and sinks. The cost of these arcs is equal to the revenue of the demand plus a penalty for each rejected container, representing a goodwill cost. The rejection penalty was set to 1000 USD after consultation with the shipping company.

Having defined the graph (N,A) we can formulate the LSNDP as follows:

$$max \sum_{k \in K} r_k d_k - \sum_{r \in R} \overline{c}_r y_r - \sum_{(i,j) \in A} \sum_{k \in K} c_{ij} x^k_{ij} \tag{1}$$

$$s.t. \sum_{k \in K} x^k_{ij} \leq u_{v_r} y_r \qquad (i,j) \in A_r, r \in R \tag{2}$$

$$\sum_{r \in R : v_r = v} y_r n_r \leq z_v \qquad v \in V \tag{3}$$

$$\sum_{j \in N} x^k_{ij} - \sum_{j \in N} x^k_{ji} = 0 \qquad i \in N \setminus \{O_k, D_k\}, k \in K \tag{4}$$

$$\sum_{j \in N} x^k_{ij} - \sum_{j \in N} x^k_{ji} = d_k \qquad i = O_k, k \in K \tag{5}$$

$$\sum_{j \in N} x^k_{ij} - \sum_{j \in N} x^k_{ji} = -d_k \qquad i = D_k, k \in K \tag{6}$$

$$x^k_{ij} \in \mathbb{Z}^+ \cup \{0\} \qquad (i,j) \in A, k \in K \tag{7}$$

$$y_r \in \{0,1\} \qquad r \in R \tag{8}$$

The objective (1) is to maximize profit, which is revenue minus costs. The first term $\sum_{k \in K} r_k d_k$ describing the revenue, is a constant and the problem can thus also be viewed as cost minimization. The next term $\sum_{r \in R} \overline{c}_r y_r$ is the network cost, while

the last term $\sum_{(i,j)\in A}\sum_{k\in K} c_{ij}x_{ij}^{k}$ is the cost of flowing the containers through the network.

The network cost is made up of the fixed costs of operating the network. This includes the capital and leasing cost of the vessels used in the network, the cost of crewing the vessels, the fuel cost, the cost of calling ports and the cost of passing the Suez and Panama canals. The flow cost is the variable cost by flowing containers through the network, which includes the cost of loading, unloading and transshipping containers at ports. As it might not be possible or profitable to transport all containers, demand can be rejected. This is associated with a revenue loss, offsetting the revenue obtained for that cargo, as well as a goodwill cost.

Constraint (2) ensures that the capacity of the arcs is respected. If a rotation is not used in the solution, the arcs that represent this rotation can of course not be used for flowing commodities. Constraint (3) ensures that at most the available number of vessels for each vessel class is used in the solution. Constraints (4), (5) and (6) ensure flow conservation for the commodities. Constraint (4) ensures that all nodes, beside the centroid nodes, have equal amount of commodity k entering and leaving. As can be seen from constraints (5) and (6) all commodities must be flown through the graph, which is possible since we have introduced omission arcs A_o. The domain of the variables is defined in constraints (7) and (8).

The considered model is a simplification of real-life LSNDP: First of all, all containers are assumed to be of the same type, hence we do not take reefer containers or high-cube containers into account. Moreover, no time constraints are present. Brouer et al. (2015) describes how these constraints can be modeled and present an algorithm for solving the resulting problem.

In addition to transit time, other complex and special constraints have been left out, for example cabotage and embargo rules that prevent certain cargo from being transported on certain vessels. The capacity at ports is unknown and assumed to be practically unlimited. For all ports, the handling costs per container is assumed to be constant regardless of the number of containers handled at the port.

3 Flowing containers

Once a set of rotations $r \in R$ have been selected, computing the optimal cargo flow can be solved as a standard multi-commodity flow problem (MCF):

$$max \sum_{k \in K} r_k d_k - \sum_{(i,j) \in A} \sum_{k \in K} c_{ij} x_{ij}^k \tag{9}$$

$$s.t. \sum_{k \in K} x_{ij}^k \leq u_{ij} \qquad (i,j) \in A \tag{10}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{ji}^k = 0 \qquad i \in N \setminus \{O_k, D_k\}, k \in K \tag{11}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{ji}^k = d_k \qquad i = O_k, k \in K \tag{12}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{ji}^k = -d_k \qquad i = D_k, k \in K \tag{13}$$

$$x_{ij}^k \in \mathbb{Z}^+ \cup \{0\} \qquad (i,j) \in A, k \in K \tag{14}$$

Although efficient algorithms exist for MCF, large instances tend to be time-consuming to solve. Brouer et al. (2015) makes use of a delta evaluation method to solve the MCF by warm-starting the search from the previous solution. Here, we use instead an even faster Lagrange heuristic (Ahuja et al., 1993). Relaxing the capacity constraints (10) using multipliers $\lambda_{ij} \geq 0$ leads to the following objective

$$max \sum_{k \in K} r_k d_k - \sum_{(i,j) \in A} \sum_{k \in K} (c_{ij} + \lambda_{ij}) x_{ij}^k + \Lambda \tag{15}$$

subject to constraints (11) to (14). The constant Λ is given by $\Lambda = \sum_{(i,j) \in A} \sum_{k \in K} \lambda_{ij} u_{ij}$. This problem can be decomposed in a number of shortest-path problems for each commodity $k \in K$, and hence solved efficiently. We use a label correcting algorithm similar to Bellman-Ford for solving the shortest-path problems.

Lagrange multipliers are updated according to the scheme

$$\lambda_{ij}^{n+1} := \lambda_{ij}^n + \begin{cases} \delta & \text{if overflow on } (i,j) \\ -\mu\delta & \text{otherwise} \end{cases}$$

To further allow fine tuning of the Lagrange multipliers, the step size is adjusted with respect to the number of iterations. In every fifth iteration, the step size is reduced by σ, which is a percentage of the current step size. Additional details can be found in Thorsen and Krogsgaard (2016).

3.1 Repair flow algorithm

For the problem at hand, it is not easy to find the convergence values for the Lagrange multipliers, as an equilibrium is almost never reached. This is due to the fact that the OD matrix contains pairs with a demand of more than 1000 containers per week. When solving the shortest-path problems for each commodity, all containers on the OD pair will follow the same path, and hence edges will either have flow 0 or the full demand (possibly violating the capacity).

In order to get a feasible flow, a repair algorithm is introduced, as outlined in Algorithm 1.

Algorithm 1 `Find Repair Flow`

```
1: set InvalidFlow := false
2: for all sail arcs s do
3:     if overflow on s is bigger than zero then
4:         set InvalidFlow := true
5:         save overflow route for s
6: for all overflow routes o do
7:     run modified Bellman-Ford for o, disregarding arcs exceeding capacity
8: if InvalidFlow then
9:     call Find Repair Flow
```

The general idea of the repair flow algorithm is to gradually turn an invalid flow, where the capacity constraints are violated, into a valid flow through a heuristic method. This is done by first finding all arcs where the load exceeds the capacity. For these arcs, the route which generates the lowest profit per container is identified. Some or all containers using this route must be moved to an alternative route to remove or reduce the overflow of the arc. The alternative route is computed with an altered version of the Bellman Ford-algorithm, where the capacity of the arcs is taken into consideration. Only arcs with spare capacity are allowed in the shortest path. The profit per container is used to decide which containers to reroute. This is done to prevent containers with high profit from ending up on an omission route, as this significantly decreases the objective value. When the profit per container is calculated, the Lagrange multipliers are included as a cost, which means that containers passing several capacity restricted arcs with high Lagrange multipliers are more likely to be rerouted than containers passing only one capacity restricted arc. The algorithm thus finds better solutions with better tuned Lagrange multipliers. The running time of the Repair Flow algorithm is dependent on the amount of overflow in the network. In a network with low overflow, it is of course less time consuming to reroute the necessary amount of containers than in a network with high overflow. To keep running times down, the Repair Flow algorithm is run only if the average, relative overflow in the network is sufficiently small.

The computation of a container flow has a time complexity of $O(A \cdot N \cdot A_s \cdot K^2)$. The Bellman-Ford algorithm runs in time $O(A \cdot N)$. Each sail arc A_s is investigated to find a commodity that can be moved to an alternative route if the load exceeds the capacity of the arc. To decide which commodity to move, the profit for all commodities using the arc must be calculated — in worst case, all commodities K will be using the same sail arc, resulting in K calculations. The algorithm is then iterated until a valid flow is obtained, which in the worst case results in K iterations, since only one commodity is rerouted per iteration.

In the sequel we will use the terminology $O(F)$ to denote the time complexity of computing the container flow.

		Heuristic		Exact method (CPLEX)		Difference	
	Number of nodes	Objective	Time (s)	Objective	Time (s)	Pct from optimality	Speed up factor
Mediterranean	684	3,723,903	1.4	3,652,740	3.7	1.9%	2.61
Pacific	816	23,757,270	1.8	22,678,697	52.0	4.8%	28,23
WorldSmall	1086	108,606,072	4.4	102,900,432	970.7	5.5%	223.04

Table 3 Results and comparisons between heuristic and exact method. The number of nodes is the number of nodes contained in the graph used for the heuristic search and the number of nodes as input data to CPLEX. The Mediterranean instance has 40 ports and 365 demands, the Pacific instance 45 ports and 722 demands and the WorldSmall instance 47 ports and 1764 demands. Pct from optimality is the gap from heuristic to optimal solution value. Speed up factor is the running time for the exact method divided by the running time for the heuristic method.

3.2 Solution times for flowing the containers

Table 3 compares the results of the Lagrange heuristic with an exact solver for the MCF. As seen, the larger instances are significantly faster to solve by the Lagrange heuristic than using CPLEX, but the solution quality has a gap of 2–5% for the larger instances. However, as the solution to the MCF is only used as a guideline for the search procedure, the gap is not considered to be a big problem. It is more important that the algorithm is consistent, i.e. that the gap is not fluctuating significantly from iteration to iteration, as an improving move might otherwise be evaluated as a deterioration and vice versa.

4 Generating initial rotations

It is important to generate a good initial solution to limit the succeeding local search. Many algorithms for LSNDP use the existing network as start solution, but this can lead to a biased solution, that does not fully exploit the solution space. If a network is designed from scratch, only the OD matrix can be used as a starting point, and it is necessary to find a method to generate rotations that takes transshipments into account. The OD pairs in themselves do not provide any information on ideal transshipment ports, so this information must be inferred in another way, for example by looking at the location of transshipment ports in the current shipping networks and use this as a guideline for generating rotations. This is done by Meng and Wang (2011), who divide the ports into hub and feeder ports. Transshipments are only allowed at hubs. For the Asia-North Europe corridor, six hub ports are defined, while all other ports are defined as feeder ports and associated with exactly one hub port. The obvious weakness of this method, however, is that it is likely to result in a network with a high degree of similarity to the current network and rule out solutions with a completely different structure. Although the current network structure is probably of high quality, it cannot be known if a different structure is even better when this option is not considered due to the method used to generate rotations.

Both Meng and Wang (2011) and Balakrishnan and Karsten (2017) suggest a method for generating a network from a list of candidate rotations. This does not solve the base problem, however, which is to construct the rotations — and a good network can of course only be constructed from a list of candidate rotations if the list contains rotations of high quality. Reinhardt and Pisinger (2012) use a branch-and-cut method to generate rotations for smaller instances of up to 10 ports, which can be solved to optimality. For a larger instance with 15 ports, however, the running time reaches the maximum limit of $5\frac{1}{2}$ hours without finding the optimal solution. This method can thus not be expected to work for a worldwide instance, where many more ports are required to get a realistic network. Plum et al. (2014) introduces the first model which allows multiple butterfly calls in a rotation. The model is solved using CPLEX, but optimal solutions are not found within the time limit for the smallest LINER-LIB instances Baltic and WestAfrica.

Brouer et al. (2014b) views the construction procedure as a variant of the multiple quadratic knapsack problem, with each rotation being a knapsack and the port calls items of the knapsack. The revenue of an OD-pair is only obtained if both ports of the pair are selected in the same knapsack. Transshipments are handled by dividing the ports into feeder and hub ports, such that demands originating or ending at a feeder port are split at a hub port. This only solves the problem of transshipments partially, however, as transshipments frequently occur in OD-pairs between two hub ports. The problem is solved heuristically, which yields relatively poor results compared to Brouer et al. (2014a) when only the construction heuristic is considered. After running the construction heuristic, the rotations are improved using other methods, leading to significantly better results.

4.1 Constructing a backbone flow

The previously implemented methods for generating the initial rotations thus suffer from long computational times, result in solutions of poor quality or require a list of candidate rotations. As there is room for improvement, a completely different approach from Pisinger (2016) is chosen to generate the initial rotations. The idea is to reverse the usual work order of first generating rotations and then flowing containers, such that containers are flown first and the rotations are then generated based on this backbone flow. As there are no rotations to flow the containers through, an artificial network must be used instead. This network is represented by a complete, directed graph, consisting of a set of nodes P, with one node for each port in the instance, and a set of arcs A connecting every node. The underlying assumption of this graph is that there is a rotation connecting all ports to each other with direct services. This network will of course never be used in practice, as it is far too expensive to run, so the arcs must be priced in a way such that the containers are grouped on a few of the arcs.

For this purpose, a non-linear multi-commodity flow problem is solved, where the objective in problem (9) to (14) is replaced with

$$max \sum_{(i,j)\in A} f(\sum_{k\in K} x_{ij}^k) \tag{16}$$

The nonlinear concave function $f(x)$ reflects economy of scale for flowing more containers: There is a large cost associated with opening an arc (i.e. deploying a vessel), while the cost per container decreases as the flow is increased. This pricing implicitly aims at aggregating the flow on fewer arcs. Sun and Zheng (2016) is also using a concave function to optimize the container flow.

For every edge (i,j) between ports i and j, the function f can be calculated by first calculating the cost a_v of sailing the leg for each vessel class $v \in V$ having capacity u_v. Next, the points (u_v, a_v) are plotted, and a curve fitting is used to approximate the points with a square-root function $f(u)$ of the capacity.

The non-linear multi-commodity flow problem is solved by repeatedly going through the following steps: Randomly choose an OD-pair with some un-routed demand, find a cheapest path (with respect to the current pricing of edges), flow a single container, and update the edge costs. Edge weights w_{ij} in the shortest path problem can be found as $w_{ij} = f(x_{ij}+1) - f(x_{ij})$ where x_{ij} is the current flow on edge (i,j).

As the arc costs depend on previously flown containers, the result of the flow will be very dependent on the order in which containers are flown. Generally, the first containers are more decisive for the arcs used heavily in the final solution than the last containers flown. It is thus necessary to run several iterations of the problem, with a random order of the containers, to achieve a reasonable *average* picture of the backbone flow. Running 10 iterations for the demand matrix of the WorldSmall instance gives the average arc loads shown in Figure 1.

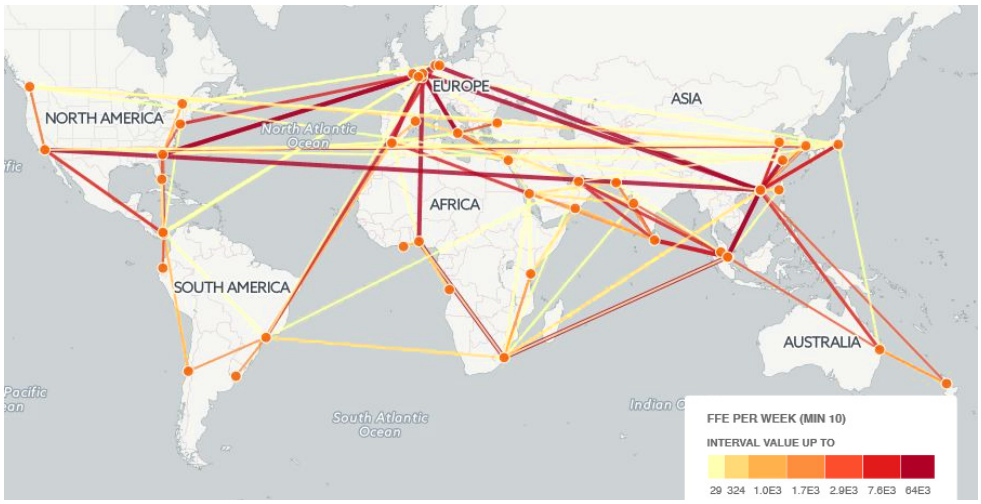

Fig. 1 Typical backbone flow for the WorldSmall instance

The figure clearly shows that only a fraction of the possible arcs is used in the solution. The number of open arcs, however, is not at a minimum, as there are many cycles in the network — it is not a tree. This can be explained by the pricing structure, as it is always connected with some cost to use an arc, even if it is already in use. In some cases, it will thus be cheaper to open a new arc, especially if a long detour can be avoided, as the cost depends on the length of the arc. It should be noted that only arcs with a load of more than 10 are included in the figure. There is a substantial number of arcs that are used but only by very few containers.

To generate good rotations based on this flow, the objective must be to cover as many of the heavily used arcs as possible. In order to formulate the problem mathematically, we use the notation defined in Table 4.

Sets	
P	Set of ports in the model
A	Set of arcs of the graph, connecting the ports
R	Set of rotations
V	Set of vessel classes
Parameters	
l_{ij}	Computed load for arc $(i,j) \in A$
d_{ij}	Distance in nautical miles covered on arc $(i,j) \in A$
p_j	Port stay at port $j \in P$. All port stays are fixed at 24 hours
t_r	Maximum duration in weeks for rotation $r \in R$
v_r	Vessel class used for rotation $r \in R$
$s_{min,v}$	The minimum sailing speed in knots for vessel class $v \in V$
$s_{max,v}$	The maximum sailing speed in knots for vessel class $v \in V$
θ^v_{ij}	(binary) 1 iff vessel class $v \in V$ is compatible with arc $(i,j) \in A$ (wrt. draft, canal, port)
Decision variables	
x^r_{ij}	(binary) 1 iff arc $(i,j) \in A$ is serviced by rotation $r \in R$
s_r	The sailing speed in knots used for rotation $r \in R$

Table 4 Notation used in model for the backbone flow

We define the set of feasible rotations R as the set of routes $r \subseteq P$ satisfying:

$$r \text{ is a connected closed loop} \tag{17}$$

$$s_r \geq s_{\min,v_r} \tag{18}$$

$$s_r \leq s_{\max,v_r} \tag{19}$$

$$\sum_{(i,j)\in A} d_{ij} x^r_{ij}/s_r + \sum_{(i,j)\in A} p_j x^r_{ij} \leq t_r \tag{20}$$

Constraints (18) and (19) ensure that the speed is set in accordance with the capabilities of the vessel used on the rotation. Constraint (20) ensures that the duration of the rotation is kept below a preset limit to avoid extremely long rotations.

This leads to the following mathematical model for constructing the rotations:

$$max \sum_{(i,j)\in A} \sum_{r\in R} l_{ij} x_{ij}^r \tag{21}$$

$$s.t. \sum_{r\in R} x_{ij}^r \leq 1 \qquad (i,j) \in A \tag{22}$$

$$\sum_{j\in P} x_{ij}^r - \sum_{j\in P} x_{ji}^r = 0 \qquad i \in P, r \in R \tag{23}$$

$$x_{ij}^r \leq \Theta_{ij}^{v_r} \qquad (i,j) \in A, r \in R \tag{24}$$

$$x_{ij}^r \in \{0,1\} \qquad (i,j) \in A, r \in R \tag{25}$$

$$s_r \in \mathbb{R}^+ \cup \{0\} \qquad r \in R \tag{26}$$

Since $r \in R$, constraints (17) to (20) are implicitly assumed to hold. The objective (21) maximizes the total load covered by rotations. Constraint (22) ensures that each arc can only be used once, preventing duplicate rotations. Constraint (23) ensures flow conservation, Constraint (24) ensures that the vessel is capable of traversing the arc, i.e. that the draft limit at both ports and any canals passed is respected. Constraint (25) and (26) define the domain of the variables.

For every rotation $r \in R$ the heuristic will choose the speed of the vessel such that it satisfies the minimum and maximum speed restrictions, and results in lowest possible operating cost. Since we have weekly operations at all ports, we need to deploy k vessels if a rotation takes k weeks. Hence the speed is a function of the number of vessels deployed. By enumerating all numbers k of vessels such that the resulting speed satisfies (18)–(19) the minimum operational costs on a rotation are easily found.

4.2 Greedy heuristic for generating rotations

A simple approach is chosen for generating rotations, as this allows quick generation of numerous candidate solutions. The idea is to add one arc at a time to a rotation until all rotations have reached their maximum duration, which can be done in two principal ways:

- Parallel generation of rotations: In every iteration, a random rotation is chosen, and the best arc for this given rotation is added. In every iteration, the unserved arc with the highest demand is added to the best possible rotation.
- Serial generation of rotations: Rotations are generated one at a time, and in every iteration, the best, unserved arc for the rotation currently being constructed is added.

The advantage of the parallel generation of rotations is that good arcs can be distributed fairly among the rotations, such that not all the good arcs are used in the rotations generated first. However, if we generate the rotations sequentially according to vessels size, the arcs with heavy load will be assigned to rotations with large vessels (exactly as expected), but restrictions on port draft will also ensure that not

all good arcs are assigned to the rotations generated first. We therefore choose a sequential generation of rotations.

For each rotation, the unserved arc with the largest flow is selected as the first arc in the rotation, and a return arc is added to close the rotation. While the rotation is at or below the desired duration, a new arc is added to the rotation to expand it, and this arc replaces the return arc. The new arc is the unserved arc with the largest demand that either starts at the same port as the return arc, which is to be replaced, or ends at the same port as the return arc. A new return arc is added to close the rotation. The selection process continues until it is not possible to add a new arc without exceeding the maximum duration of the rotation. After this, the creation of the next rotation starts.

To obtain a number of different start solutions to select from, the algorithm is repeated a number of times with random settings on the maximum rotation length for every rotation. The length is selected in a predefined interval depending on the size of the vessel, such that larger vessels, typically travelling between continents, get longer rotations than smaller vessels doing feeder service. To allow some variation, wide intervals are selected for most vessel classes as shown in Table 5.

Vessel class	Rotation length
7500 FFE (Super Panamax)	[10,10] weeks
4200 FFE (Post Panamax)	[7,14] weeks
2400 FFE (Panamax)	[6,12] weeks
1200 FFE (Panamax)	[4,10] weeks
800 FFE (Feeder)	[2,8] weeks
450 FFE (Feeder)	[2,5] weeks

Table 5 Allowed duration of rotations based on vessel class in the rotation generation procedure

For the largest vessel class either 0 or 10 vessels are available in all base instances, and as these vessels must be used on intercontinental routes with high volume, it does not make sense to split the vessels on two different rotations. One rotation of 10 weeks must thus be constructed (if the vessel is available in the given instance). For the other vessel classes, intervals that have shown to generate reasonable results have been selected. For every rotation generated, a duration is selected in the interval at random, and the rotation is constructed. This is repeated until all available resources have been exhausted.

4.3 Results for generation of initial rotations

To evaluate the effectiveness of the algorithm, tests are conducted on the WorldSmall case. The flow is computed five times with each flow being an average over ten iterations, where the containers are flown in random order. For each of the five flows, 20 different sets of rotations are created. This yields the solution values shown in Table 6.

	Best solution value	Median solution value	Worst solution value	Running time (sec)
Flow-1	6.56	-2.83	-18.06	76.19
Flow-2	11.04	-4.29	-15.14	79.32
Flow-3	11.91	3.94	-14.04	78.77
Flow-4	0.92	-12.63	-26.84	80.05
Flow-5	17.22	-2.39	-18.66	81.57

Table 6 Resulting objective values (in M$) for 20 different sets of initial rotations for each of five different backbone flows.

As can be seen from the table, the results vary considerably between the five different flows. In general, Flow-4 gives the poorest results with the best solution being just barely profitable, while the best solution overall is found with Flow-5. Flow-3, however, has the best solution value for the median and worst solutions, even though the solution value for the best solution is significantly lower than for Flow-5. The best known solution to the WorldSmall case has an objective value of 58 M$ per week, so even though all flows generate a profitable solution, the solutions are still far from optimal. There is also a big variation between best and worst solutions for each flow, and the random selection of rotation duration thus has a significant impact on the objective value.

The running time is very stable at around 80 seconds, which is encouraging: Such short computational time allows generation of a high number of different start solutions within a reasonable time limit.

5 Network optimization

With the initial rotations generated, the network must be optimized to get a high quality solution.

We have chosen to use Variable Neighborhood Search (VNS), described by Hansen and Mladenovic (2014). The general idea is to apply different neighborhood structures throughout the search to exploit the benefits from neighborhood changes. When a local optimum is encountered, it is escaped by doing a random move, a `shake`, from the best known solution and do hill climbing from here until a new local optimum is reached. If this solution is better than the previously best known, the search is continued from here with a new shake, otherwise the search returns to the previously best known solution and searches from here again after a new shake. The pseudocode of the metaheuristic is given in Algorithm 2.

As can be seen from the pseudocode, the `local search` procedure terminates after all neighborhoods have been tested without yielding an improving solution, as a local optimum with respect to all neighborhoods must then have been encountered. The `shake` procedure is applied less frequently than in a standard VNS framework. A lower degree of randomness is preferable here, because the evaluations are rel-

Algorithm 2 `Improvement Algorithm`

```
1: Initialization: Find an initial solution x
2: while stopping criterion not met do
3:     generate a new solution x' from x (shake)
4:     while any neighborhood in N is unused (local search) do
5:         choose at random an unused neighborhood and search from x'
6:         if an improved solution x'' is found then
7:             Set x' := x'' and set all neighborhoods unused
8:         if x' is better than x (move or not) then
9:             Set x := x'
10: return x
```

atively expensive. It is thus desirable to search directly for a local optimum with respect to all neighborhoods before randomly altering the solution.

Although the local search only accepts moves that have an expected improvement, some moves may turn out to be degrading when calculating the real objective function. These moves are nevertheless kept on to progress the search. This can, however, lead to cycling in the local search, as it might both be expected to be an improvement to first insert a port and to remove it afterwards. To break such cycles, only 20 loops are allowed in the `local search` part, after which the algorithm must continue to `move or not`.

If a cycle is encountered or a local optimum has been reached, the `shake` procedure is applied to progress the search from another point in the solution space. The procedure must change the solution sufficiently to escape the local optimum, but should, on the other hand, not destroy good characteristics of the solution. Preliminary studies show that there is a high risk of changing the solution too much to be able to return to a good solution, and a relatively modest shake procedure is thus implemented. This procedure modifies a number of rotations by either inserting or removing a port randomly, without considering the effect on objective value. To avoid inserting an obviously irrelevant port, a distance requirement is enforced such that only ports relatively close to the rotation can be inserted. The number of modified rotations is 10% of the total number of rotations and a least one.

In each iteration of `local search`, a neighborhood is randomly selected and one or more rotations are altered through that neighborhood. Six different neighborhoods are applied:

- Insert port
- Service omission
- Service unserved port
- Remove port
- Simple remove port
- Create feeder rotation

In order to select the best move, delta evaluation is used to avoid time consuming evaluations of the multicommodity flow for the entire graph. Instead, a small graph (*rotation graph*) is constructed, covering only the rotation currently being altered.

As this graph is much faster to evaluate, more moves can be tested before one is selected for implementation.

In the following, the rotation graph and the six neighborhoods are described in more detail. Additional implementation details can be found in Thorsen and Krogsgaard (2016).

5.1 Rotation graph

The rotation subgraph covers only a single rotation r. It has the same structure as the main graph, which was described in Section 2, with some important alterations. A special demand matrix is constructed for the rotation graph, containing only demand using rotation r in the main graph. As transshipments are very common, origin and destination for some of this cargo will be at ports not serviced by r. When only r is included in the rotation graph, however, it is only possible to route this cargo on omission arcs, as either origin or destination or both will be without a connection to r.

As this is not desirable, feeder arcs A_f are introduced to the rotation graph structure. Where the arc set of the main graph is $A = A_s \cup A_t \cup A_l \cup A_u \cup A_d \cup A_o$ (sail, transshipment, load, unload, dwell and omission arcs), the arc set of the rotation graph is $A = A_s \cup A_l \cup A_u \cup A_d \cup A_o \cup A_f$. The set of feeder arcs can be subdivided further into three subsets $A_f = A_{f,t} \cup A_{f,l} \cup A_{f,u}$, which are defined as follows: $A_{f,t} = \{(i,j) \mid i \in N_{arr}^{p_1}, j \in N_{dep}^{p_2}, p_1, p_2 \in P, p_1 \neq p_2\}$, $A_{f,l} = \{(\alpha^{p_1}, i) \mid i \in N_{dep}^{p_2}, p_1, p_2 \in P, p_1 \neq p_2\}$ and $A_{f,u} = \{(i, \beta^{p_2}) \mid i \in N_{arr}^{p_1}, p_1, p_2 \in P, p_1 \neq p_2\}$. The feeder arcs can thus represent either:

- Cargo transshipping from r to another rotation at port p_1 and back to r at port p_2, represented by set $A_{f,t}$. Although uncommon, such routes do occur.
- Cargo transshipping from another rotation to r at port p_2, represented by set $A_{f,l}$. These arcs lead the cargo directly from its source node α^{p_1} to the transshipment port p_2.
- Cargo transshipping from r to another rotation at port p_1, represented by set $A_{f,u}$. These arcs lead the cargo directly from the transshipment port p_1 to the sink node β^{p_2} of the cargo.

Figure 2 shows how a main graph with three rotations is transformed into a rotation graph for the rotation r marked in red. For ease of explanation, transshipment, load and unload arcs have been left out of the illustration. Only a very limited number of feeder arcs are necessary to replace the rotations indicated in grey.

The feeder arcs are generated from the commodity routes used in the main graph. Any commodity route using rotation r can cause several feeder arcs to be created. Only one feeder arc can exist between two given nodes, so if any of the feeder arcs already exist, they are not created again. The feeder arcs are capacitated to keep as much structure from the multicommodity flow of the main graph as possible. For any feeder arc, the capacity is calculated as follows:

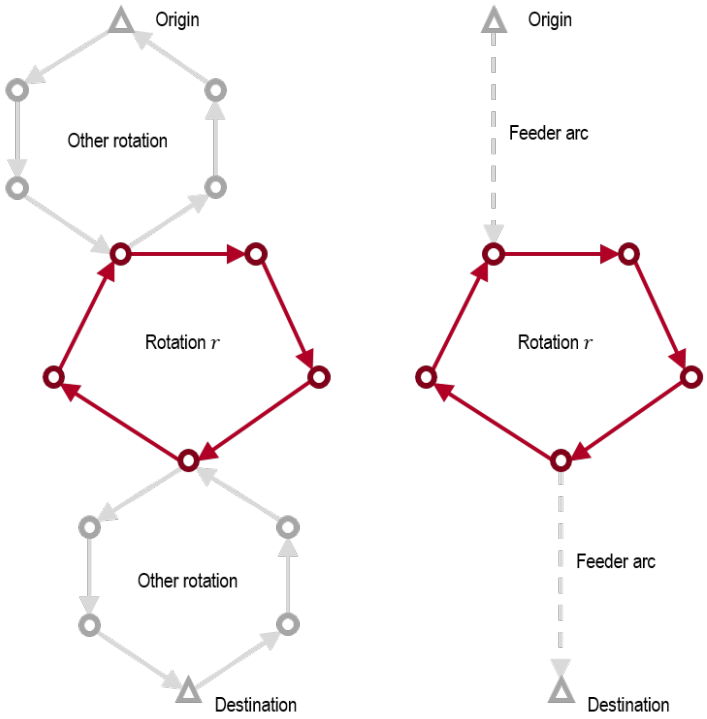

Fig. 2 Conversion from main graph to rotation graph. Main graph (left) and transformed rotation graph (right).

$$u_{ij} = \sum_{k \in K} \tilde{x}_{ij}^{k} + \tilde{u}_{ij}$$

where $\tilde{x}_{ij}^{k}$ is the number of commodity k using arcs corresponding to feeder arc (i, j) in the main graph and $\tilde{u}_{ij}$ is the spare capacity of the most restricting of the arcs of the main graph corresponding to feeder arc (i, j). It depends on the local search neighborhood if omission cargo is included.

The feeder arcs are associated with a cost to provide an incentive for using rotation r as much as possible, i.e. to prefer short feeder arcs over long feeder arcs if a choice is available. The cost also ensures that there is an incentive for replacing heavily loaded feeder arcs with an extension of rotation r. costs are calculated according to the following equation:

$$c_{ij} = \frac{c_b^{v_r} + c_{TC}^{v_r} + c_c^{v_r} + c_i^{v_r} + c_j^{v_r}}{u_{v_r}} + c_{h_i} + c_{h_j}$$

where v_r is the vessel class used in rotation r, $c_b^{v_r}$ is the bunker cost for sailing at design speed, $c_{TC}^{v_r}$ is the capital cost for the vessel, $c_c^{v_r}$ is the cost for the canals

passed on the arc, $c_i^{v_r}$ is the cost for calling at port i, $c_j^{v_r}$ is the cost for calling at port j and u_{v_r} is the capacity of the vessel. c_{h_i} and c_{h_j} are the handling costs per container at ports i and j, respectively, and it depends on the type of feeder arc if load/unload or transshipment costs are used. If the feeder arc connects from a centroid to a transshipment point, the load cost is used at port i while the transshipment cost is used at port j.

5.2 Insert port

The insert port-neighborhood tries to insert a port into a rotation r. The method makes use of the rotation graph, which is constructed for r. In addition to the demand serviced by r, omission demand is added to the graph as well. This demand must meet two requirements to be included in the graph:

- Either its origin or its destination must be at a port serviced by r.
- The origin or destination port not serviced by r must be part of the rotation graph by connection with a feeder arc.

New feeder arcs are not created to allow omission demand to be included. As the existing feeder arcs all represent other rotations, the method will thus never be able to include omission demand at a port not served by any rotation at all. This is handled by the service omission-neighborhood described below.

The neighborhood works by testing insertion of a port connected to the rotation by a feeder arc by replacing the feeder arc with an extension of rotation r. The feeder arc must, however, fulfill the following requirements to be taken into consideration:

- Its capacity must be fully utilized. If there is spare capacity on the feeder arc, it is very unlikely that it is profitable to add further capacity by extending r.
- The load on the arc, i.e. the number of containers transshipping to or from r, must be at least 5% of the capacity of the vessel used in r. If the load is less, it is very unlikely that will be profitable to replace it with an extension of r.

If these two requirements are met, it is tested if the feeder arc can be profitably replaced by an extension of rotation r, such that the port served by the feeder arc becomes part of r. For a feeder arc leading from port p to port j in rotation r, the rotation is extended by inserting p before j in r. For a feeder arc leading to port p from port i in rotation r, the rotation is extended by inserting p after i in r.

The time complexity of this neighborhood is $O(A_f \cdot F)$, where $O(F)$ is the time required for evaluating the flow. The neighborhood investigates all feeder arcs A_f in the graph to find a profitable insertion. Each insertion possibility is evaluated with the algorithm described in Section 3.

5.3 Service omission

The service omission-neighborhood seeks to improve service at ports having a large amount of cargo that cannot be routed through the network and thus flows on omission arcs, contributing negatively to the objective value. The general idea is to first identify the port at which service is to be improved and then test insertion of the port in the rotation graph of every rotation. The port $p_{omission}$ is identified by calculating the total omission demand, including both origin and destination cargo, for every port in the graph and randomly select one of the five ports having the largest omission demand. For every rotation, the port in the rotation having the shortest distance to $p_{omission}$, $p^r_{closest}$, is found, and $p_{omission}$ is inserted into the rotation in an out-and-back fashion.

The time complexity of the neighborhood is $O(P^3 \cdot R \cdot F)$, where $O(F)$ is the time required for evaluating the flow. The neighborhood first investigates all ports P to identify the one where service is to be improved. This involves the calculation of omission demand for each port, which requires going through potentially $2(P-1)$ different commodities — one to and from each other port in the dataset. The complexity of finding the omission port is thus $O(P^2)$. All rotations must then be tested for insertion of the omission port with the shortest possible extension of the rotation, which in the worst case requires running through all ports in all rotations, $O(R \cdot P)$, to find the optimal insertion point.

5.4 Service unserved port

The service unserved port-neighborhood seeks to introduce service to unserved ports. This is not captured by the insert port-neighborhood, as it tests insertion of ports into a rotation only if the port is connected with a feeder arc. As feeder arcs represent existing rotations and an unserved port obviously has no rotations calling at it, an unserved port will not be inserted by the insert port-neighborhood. It can, however, be inserted by the service omission-neighborhood, but as this neighborhood considers only the five ports generating most omission demand, smaller ports are not likely to be captured by it.

The service unserved port-neighborhood selects a port between all ports currently not serviced in the solution. The probability of selecting a port is equal to its proportion of the demand of all unserved ports, such that an unserved port with high demand is more likely to be selected than an unserved port with low demand. Apart from the selection of ports, the neighborhood works exactly like the service omission-neighborhood described above and can thus be expected to predict improving moves just as accurately — although probably with a lower average improvement, as the ports included in this neighborhood have less omission demand than the ones included in the service omission-neighborhood.

As the only difference from the neighborhood described in Section 5.3 is the criteria for selecting the insertion port, which does not affect the complexity, the

time complexity is exactly the same $O(P^3 \cdot R \cdot F)$, where $O(F)$ is the time required for evaluating the flow.

5.5 Remove port

The remove port-neighborhood tries to remove a port call from rotation r. The call must either be one of more calls at the same port by r or have a relatively low exchange of cargo to be taken into consideration. If it is one of more calls, it will often be possible to profitably remove it, as the cargo can be serviced by the other calls of r at the same port. If the exchange of cargo is low, the costs of calling at the port can be higher than the revenue of the serviced containers. A low exchange of cargo is defined as 30% of the vessels capacity, while the exchange of cargo is defined as loaded plus unloaded plus transshipped containers. A vessel with a capacity of 1200 FFE must thus exchange at most 360 FFE at a port call if it is to be a candidate for removal (provided that the call is the only one for the rotation at that port).

A port call q between two port calls i and j is simply removed by replacing arcs (i,q) and (q,j) with a new arc (i,j).

The time complexity of this neighborhood is $O(N_{dep} \cdot F)$, where $O(F)$ is the time required for evaluating the flow. The neighborhood investigates all port calls, equal to a departure node N_{dep} in the graph, to find a profitable port call to remove. Each removal possibility is evaluated with the algorithm described in Section 3.

5.6 Simple remove port

In view of the rather inadequate results provided by the remove port-neighborhood, a simple remove port-neighborhood is applied as a supplement. This neighborhood seeks to remove all port calls loading and unloading less than $x\%$ of the vessels capacity, i.e. to remove port calls serving relatively few containers. This limit is set at 5%, such that a port call is removed if the combined number of containers loading and unloading is less than 5% of the vessels capacity. There must, however, be more than one rotation calling at the port for a call to be removed as it will otherwise be impossible to serve ports with demand lower than 5% of the capacity of the smallest vessel.

The time complexity is $O(N_{dep})$, as the neighborhood investigates all port calls, equal to a departure node N_{dep} in the graph, to find calls which can be profitable to remove. Moves are not evaluated before being carried out, hence the stated complexity.

5.7 Create feeder rotation

Especially for the world and trade lane instances, it is vital for the quality of the network to have a well-functioning feeder network. The feeder rotations must connect two types of feeder ports to a nearby transshipment port: Ports with low demand, where it is not economical to call with large vessels. Ports with low draft, where it is not possible to call with large vessels.

It is difficult to discover good feeder rotations with the previously described neighborhoods, as they rely heavily on the demand matrix. A special neighborhood is thus used for creating feeder rotations between a feeder port and a hub port. A feeder port is defined as a port allowing only the two smallest vessel classes to enter, while a hub port is defined as a port allowing at least a medium-sized vessel of 2400 FFE to enter. Of all feeder ports, the one with the highest amount of omission demand is selected. This port must be connected to a hub port, which is selected randomly between all hub ports being within double distance of the nearest hub port. It is thus not always the nearest hub port, which is selected, as this port might not necessarily be the best fit with the network. When a feeder port and a hub port have been selected, a new rotation is constructed between the two.

The time complexity is $O(P^2)$, since the neighborhood investigates all ports P to find one lacking feeder service. All ports are then investigated again to find a suitable hub port to connect to. As moves are not evaluated before being carried out, we get the stated complexity.

6 Computational results

The algorithm has been implemented in Java and tested on the base version of the seven instances from LINER-LIB. A detailed parameter tuning can be found in Thorsen and Krogsgaard (2016). The parameter tuning also documents that all six neighborhoods used in the VNS algorithm in Section 5 contribute to the solution process, and that the applied estimate functions actually give a fair prediction of the real objective function. In the computational experiments, the backbone flow described in Section 4.1 is constructed using a slightly simplified scheme as described in Thorsen and Krogsgaard (2016).

For every instance, 5, 10 or 20 replications have been run. The maximal running time per replication has been set to 10% of the maximal running time used by Brouer et al. (2014a).

In Table 7, best, median and average objective values are reported for the seven instances. A profitable best solution is obtained for all instances except the Mediterranean instance. For average and median objective values, the Baltic and Pacific instances are also negative. Generally, the median and average objective values are close to each other, indicating that the objective values follow a non-skewed distribution around the average.

Instance	Best objective value (M$)	Median objective value (M$)	Avg. objective value (M$)	Replications	Running time (sec)
Baltic	0.09	-0.14	-0.15	10	30
WestAfrica	5.62	5.19	5.15	10	90
Mediterranean	-2.02	-2.74	-2.84	20	120
Pacific	1.83	-1.41	-1.26	20	360
AsiaEurope	25.99	20.27	20.57	10	1440
WorldSmall	50.72	43.42	43.46	20	1080
WorldLarge	25.41	13.80	15.73	5	3600

Table 7 Objective value statistics for the algorithm. The running time is per replication. Objective values are reported in M$ per week

The following tables summarize the most important properties of the best solution obtained for all seven instances. The first is Table 8 showing statistics on the performance of the heuristic.

Instance	Iterations	Improving iterations	Last improving iteration	Avg. time per iteration (sec)
Baltic	85	6	50	0.4
WestAfrica	44	15	16	2.0
Mediterranean	51	16	48	2.4
Pacific	71	30	63	5.1
AsiaEurope	105	44	89	13.7
WorldSmall	109	28	108	9.9
WorldLarge	105	48	90	34.3

Table 8 Algorithmic performance for the best solutions

Due to the different running times applied, the number of achieved iterations is of the same order of magnitude for all instances. The average time per iteration is vastly different, though, with almost a factor 100 of difference between the smallest and the biggest instance. Smaller instances generally have fewer improving iterations than the bigger instances, which is probably a result of a simpler search landscape. The progress of the search can best be described by mapping the current and best found objective values with the running time. Two examples of this are shown in Figure 3 for the Baltic instance and in Figure 4 for the WorldSmall instance.

The search progress is very different for the two instances. For the Baltic instance, the search proceeds rapidly to an objective value very close to the best found. The search then tends to cycle until a random move breaks the cycle. Cycling does not occur for the WorldSmall instance as it is much bigger and thus has more potential moves, leading to less risk of two following moves cancelling each other.

The MCF is only solved heuristically with the algorithm applied in this paper. The objective value of the best solution can thus be improved by finding an optimal solution to the MCF. This is shown in Table 9 along with a comparison with the results obtained by Brouer et al. (2014a,b).

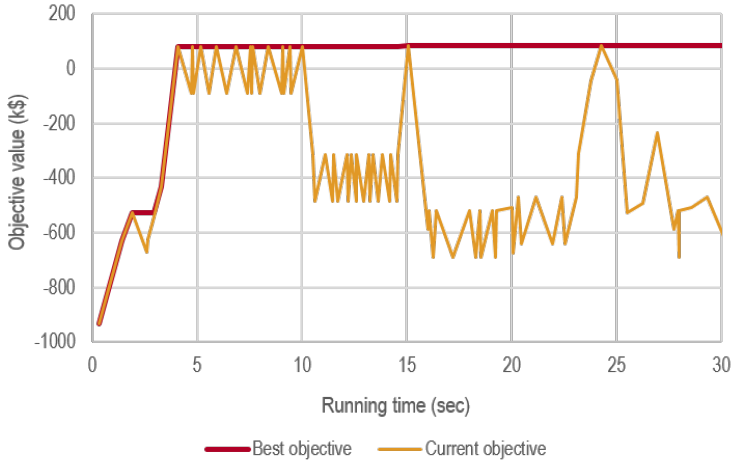

Fig. 3 Development of objective value in the Baltic instance

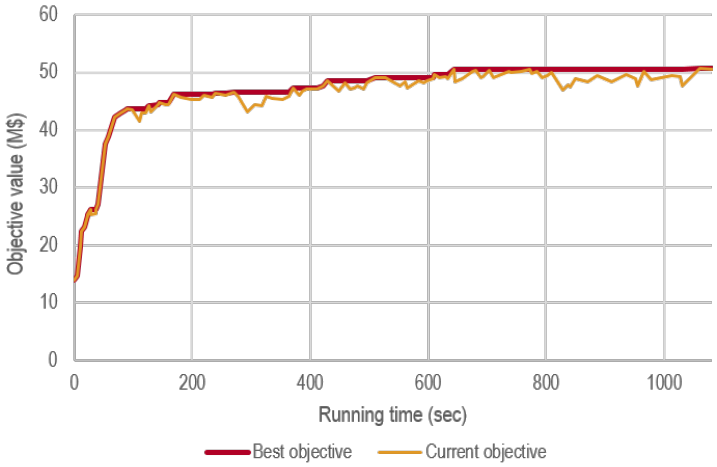

Fig. 4 Development of objective value in the WorldSmall instance

Since Brouer et al. (2014a,b) are minimizing expenses, where we are maximizing profit we also convert our solutions to the minimization form in Table 10. If z_B is the objective function by Brouer, and z_H is the objective function of the present heuristic, then the objectives have the relation $z_H = -7 \cdot Z_B/180$, since Brouer is reporting expenses per 180 days, while we are reporting weekly profits.

As can be seen from the table, the algorithm perform differently for the smaller instances. Brouer et al. (2014a) finds the best solution for Baltic and Mediterranean, while Brouer et al. (2014b) finds the best solution for Pacific and AsiaEurope, and

Instance	Objective, heuristic MCF	Objective, optimal MCF	Objective, Brouer et al. (2014a)	Objective, Brouer et al. (2014b)
Baltic	0.09	0.09	**0.33**	0.24
WestAfrica	**5.62**	**5.62**	5.57	5.44
Mediterranean	-2.02	-1.95	**-0.47**	-1.25
Pacific	1.83	2.14	2.10	**2.71**
AsiaEurope	25.99	28.36	25.59	**31.77**
WorldSmall	50.72	**58.14**	44.83	54.44
WorldLarge	**25.41**	—	—	—

Table 9 Comparison of objective values (max weekly profit) for the best solutions (in M$). The best overall solution is indicated in bold. The optimal MCF has been computed with CPLEX. The WorldLarge instance is too large for CPLEX to solve with the available computer.

Instance	Objective, heuristic MCF	Objective, optimal MCF	Objective, Brouer et al. (2014a)	Objective, Brouer et al. (2014b)
Baltic	-2.31	-2.31	**-8.37**	-6.16
WestAfrica	**-144.51**	**-144.51**	-143.11	-140
Mediterranean	51.94	50.14	**12.21**	32.2
Pacific	-47.06	-55.03	-54.09	**-69.8**
AsiaEurope	-668.31	-729.26	-657.97	**-817**
WorldSmall	-1304.23	**-1495.03**	-1152.76	-1440
WorldLarge	**-653.40**	—	—	—

Table 10 Comparison of objective values (min operational costs per 180 days) for the best solutions (in M$). The best overall solution is indicated in bold. The optimal MCF has been computed with CPLEX. The WorldLarge instance is too large for CPLEX to solve with the available computer.

the here presented algorithm finds the best solution for WestAfrica. For the world instances, however, the here presented algorithm outperforms both previous algorithms, and it is the first algorithm to report results on WorldLarge.

The number of replications is of the same order of magnitude — Brouer et al. (2014a) reports results for 10 replications for all instances. As the running time of the algorithm is set to only 10% of the running time used by Brouer et al. (2014a,b), the total time required for finding the solutions shown here is much shorter. Even when including the time for computing the initial flow, which is not included in the running time shown above, and the time for solving the final MCF through CPLEX, it is still possible to do at least eight times more replications in the same timeframe as applied by Brouer et al. (2014a,b).

Brouer et al. (2014a,b) do not solve the WorldLarge instance due to its size and we have not been able to find any other literature presenting a solution to this instance. The heuristic approach applied here is thus capable of solving a larger problem instance than other methods previously applied. As there are no other solutions to WorldLarge to compare with, it is hard to tell if the presented solution is *good*, although it can be concluded that it is profitable. Compared to the WorldSmall instance, however, it can be expected that WorldLarge will result in less profit as the demands are spread across four times as many ports, while the total demand is approximately the same and the average revenue per container is only 5% higher. The

obtained best solution to WorldLarge, which has a profit approximately half that of the best solution to WorldSmall, is probably of reasonably good quality.

The algorithm performs poorest for the Mediterranean instance. This can possibly be explained by the weekly frequency requirement enforced here, as the best solution found by Brouer et al. (2014a) contains a rotation with biweekly frequency. The Mediterranean instance contains many ports with low demand and high port call costs, hence biweekly operation is advantageous.

Instance	Deployed capacity	Rotations	Avg. rot. length (1000 nm)	Avg. port calls per rot.	Avg. rot. speed (kn)	Average LF (ACM/RCM)
Baltic	100%	2	4.55	7.00	13.54	73%
WestAfrica	97%	9	7.15	4.22	10.77	64%
Mediterranean	100%	5	5.31	9.00	11.63	75%
Pacific	100%	14	12.36	7.71	12.16	80%
AsiaEurope	93%	30	9.48	6.53	12.17	70%
WorldSmall	100%	36	13.39	5.69	12.51	76%
WorldLarge	77%	72	9.12	5.19	12.24	71%

Table 11 Properties describing the rotations of the best solutions. The deployed capacity is calculated as available vessel capacity divided by deployed vessel capacity, both measured in FFE. The average load factor is calculated as Available Container Miles (ACM) divided by Revenue Container Miles (RCM), i.e. the number of FFE miles of capacity offered divided by the number of FFE miles actually travelled by the containers.

The best solutions found by our algorithm are studied in more details in Table 11, which describe the rotations. Generally, the capacity deployment is very high with deployment of all available vessel capacity for four instances and almost all available vessel capacity for the WestAfrica instance. Only the AsiaEurope and WorldLarge instances have some spare capacity. The high degree of vessel deployment is caused by the low speeds of around 12–13 knots used in the rotations, as a low speed is, all other things being equal, requiring more vessels than a high speed. With a minimum speed of 10 knots for the feeder vessels and 12 knots for all other vessels, the slow steaming concept is taken to extremes here, which can be explained by the way the optimal speed is calculated. Only the direct operating costs are included in the calculation, such that the speed becomes a trade-off between bunker fuel costs and fixed, daily vessel operating costs (crew, capital etc.), and of these two costs, the bunker fuel cost is apparently dominant. The extra revenue, which can be obtained by increasing the speed of a rotation and e.g. add an extra port call, is not considered.

The number of rotations is, not surprisingly, closely related to the size of the instance. The average rotation length is highest for the WorldSmall instance and significantly larger than for the WorldLarge instance, which can be explained by more of the short feeder rotations for WorldLarge due to a higher number of ports in the instance. The average length is shortest for the Baltic instance due to the small geographical area covered. The average number of port calls per rotation is seemingly not correlated to either instance size or type.

The average load factor is around 75% for all instances except WestAfrica, which has a somewhat lower average load factor of 64%. It is not possible to achieve a 100% load factor due to demand imbalances, where more cargo flows from Asia to Europe and America. than the other way around, leading to half-empty vessels on the backhaul to Asia. This phenomenon also affects the feeder instances, as more cargo flows from the hub port to the feeder ports than back to the hub port. With this in mind, the utilization of the networks is at an acceptable level.

Instance	Rejected cargo	Cargo delivered within time limit	Cargo delivered on direct service	Avg. no. transshipments
Baltic	8%	100%	100%	0.00
WestAfrica	9%	67%	71%	0.29
Mediterranean	17%	60%	55%	0.43
Pacific	12%	43%	61%	0.42
AsiaEurope	15%	32%	30%	0.91
WorldSmall	17%	27%	36%	0.76
WorldLarge	18%	21%	18%	1.14

Table 12 Statistics on the cargo flow of the best solutions. Rejected cargo is cargo flown on omission arcs, while the rest is shipped through the network. The realized transit times for the shipped cargo have been compared with the time limits specified in LINER-LIB to obtain an overall percentage of cargo delivered timely. A transshipment is assumed to take four days, covering the expected waiting time with weekly services and time for unloading and loading. All statistics are based on the heuristic solution to the MCF.

Another way to describe the best solutions is to investigate the obtained cargo flow. This is studied in Table 12. The amount of rejected cargo is lowest for the single-hub instances and highest for the WorldLarge instance. The high rejection rate for WorldLarge is not caused by lack of vessel resources, as spare capacity is available, but rather a suboptimal composition of the vessel fleet. 18.0% of the total demand originates or ends at a port that can only be serviced by the smallest feeder vessels, but these vessels make up merely 1.6% of the total capacity of the fleet. All of these vessels have been deployed in the solution. Even if the shallow ports are connected to the nearest possible transshipment port at the highest possible speed, it seems unlikely that it is possible to service all demand with such a small feeder fleet. For most other instances, all vessel resources have been exhausted, and it is thus impossible to service all demand with the selected sailing speeds.

Travel time is not considered in the solution of the multicommodity flow problem, and it is thus interesting to see if the resulting solution is attractive from a travel time point-of-view. The results are generally very poor on this parameter, with up to 79% of the cargo delivered too late with respect to the time limit. For several of the instances, more cargo is delivered on a direct service than within the time limit, indicating that the sailing speeds are simply too low to meet the travel time requirements. It is even possible to find examples where a direct, non-stop service is too slow, for example in the solution to the WorldSmall instance.

Transshipments are not penalized in the objective value except for the direct cost associated with it. As transshipments are undesirable for the shippers due to risk

of damage and delay, they should generally be limited. According to Maersk Line (Plum, 2015) about half of the cargo is delivered on direct services. In our solutions it is only 20-40% of the cargo in the large problems that is delivered on a direct service. This might, though, also be related to the scarcity of vessel resources caused by low sailing speeds, as a high number of direct connections generally requires many port calls per rotation, which in turn increases the rotation time and number of deployed vessels.

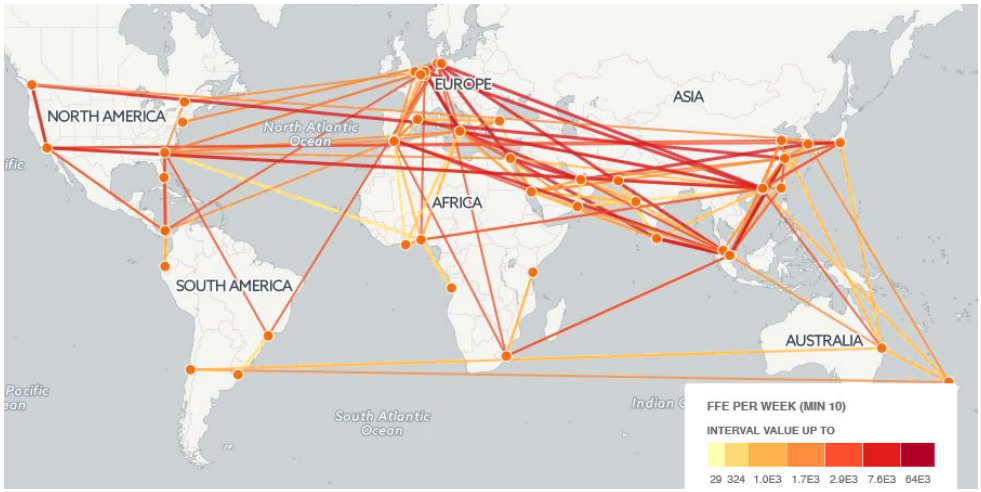

Fig. 5 Best found rotations and flow for the WorldSmall instance

Finally, it is interesting to compare the backbone flow described in Section 4.1 with the final flow. For WorldSmall the backbone flow was shown in Figure 1 while the final flow is shown in Figure 5. It is clear that the *real* flow in Figure 5 has a much more diverse structure than the backbone flow in Figure 1 with many more active arcs and typically more moderate loads on all arcs. The backbone flow has only a few active arcs, and there is typically either an extremely high load or a relatively low load on the active arcs. The backbone flow is thus not that good an approximation of a realistic flow since too few arcs are active. On the positive side, though, the arcs that are found to be attractive in the backbone flow are generally also used considerably in the real flow. If time constraints were taken into account when constructing the backbone flow, one would probably get a more diverse flow closer to reality.

7 Conclusion

The goal of this paper has been to develop a new approach for solving the Liner Shipping Network Design Problem (LSNDP) by applying heuristic methods. This includes the solution of the multicommodity flow problem, which is quite time consuming to solve to optimality for larger problems. The developed Lagrange heuristic is up to two orders of magnitude faster than an exact algorithm, finding solutions 2–5% from optimum. The memory usage is also lower, allowing the algorithm to solve larger instances that could not be solved previously. Although the solutions are not optimal, they are sufficiently good to guide the local search in the right direction.

A start solution is generated by solving a relaxed problem in which vessel capacities are "linearized" and route constraints are removed. The resulting backbone flow shows which arcs are attractive, and which ports are well suited for hubs. It is reasonably fast to compute the backbone flow and thus possible to generate many candidate solutions as a starting point. The objective values of the obtained networks are far from optimal, but the backbone flow contains valuable structural information that can be used to guide the following flow. The backbone flow can also be seen as a *physical internet* transportation network consisting of distributed multi-segment flows (Montreuil, 2011).

The initial network is optimized using a VNS-inspired heuristic using six different neighborhoods. Of these six neighborhoods, four make use of a rotation subgraph to quickly be able to delta evaluate moves before they are implemented. The subgraph includes only the rotation currently being modified, which means that the delta evaluation of a move in the subgraph cannot be expected to be fully coherent with the actual effect of implementing the move in the main graph. However, Thorsen and Krogsgaard (2016) showed that moves, which are expected to improve the solution, most often actually do improve the solution. On average, the improvement is larger than expected. In Thorsen and Krogsgaard (2016) it has furthermore been confirmed that all six neighborhoods in combination contribute positively to the objective value.

Computational results of the overall algorithm demonstrate that it provides promising results. A comparison with the results obtained by Brouer et al. (2014a) shows better solutions in four out of six instances. Furthermore, the results have been obtained over a significantly shorter running time, as the time per replication has been set to only a tenth of that used by Brouer et al. (2014a). For the seventh and largest instance, WorldLarge, no previous results have been published, and the solution presented here is thus pioneering. There is potential for improving the solutions, especially on parameters that have been excluded from the problem definition. With an improved solution to several of the instances, a novel solution to the largest instance and shorter running times, the heuristic approach adopted in this paper has great potential with regards to solving large-scale real-life LSNDP.

7.1 Future work

The most important extension of the algorithm is to include time constraints on the delivered cargo. This will mean that the backbone flow constructed in Section 4.1 should take time into account, hopefully avoiding that all flow is aggregated on very few arcs. This will form a better basis for constructing rotations. In the VNS algorithm described in Section 5 it will be necessary to use a more intelligent choice of sailing speeds, selecting higher average speed. In order to flow the containers in each iteration, it will be necessary to solve a time-constrained multi-commodity flow problem. Karsten et al. (2015) showed that although the problem theoretically becomes much harder to solve, many partial solutions can be excluded due to the time constraints, and hence the overall running times are not much larger than without time constraints.

The local search heuristic has room for improvement in several areas. First of all, it is unfortunate that two of the six neighborhoods are based solely on *rules of thumb* instead of a solid delta evaluation, as this can easily lead to degrading moves. These moves are then reversed by other neighborhoods, which in some cases causes cycling in the local search procedure. It might thus be a good idea to use a tabu list in the local search, such that immediate reversal of moves is prevented. Another solution is to reverse moves which turn out to degrade the objective value, but it will probably require a tabu list to do this too, as a reversed move will otherwise be suggested over and over again by the algorithm.

When inspecting the results, the least successful parts seem to be related to transshipment of cargo. For instances with a very fragmented demand matrix, like in WorldLarge, it is difficult for the algorithm to detect and properly service hub ports if they do not have a great amount of demand themselves — or perhaps no demand at all. An interesting extension of the algorithm could thus be a new neighborhood, where demands are split at relevant hub ports to get a more focused search towards a hub-and-spoke structure. It should be possible to do this dynamically through the rotations graphs without *hardcoding* a transshipment port for each demand pair.

Instead of having time-constraints on the delivered cargo, one could instead use transit time-dependent revenues or demands to reflect that there are customers for almost any product as long as the price is low enough — and, correspondingly, more customers and higher revenue to be gained from an attractive product with short transit times.

References

Richa Agarwal and Özlem Ergun. Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science*, 42(2):175–196, 2008.

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

Jose F. Alvarez. Joint routing and deployment of a fleet of container vessels. *Maritime Economics & Logistics*, 11:186–208, 2009.

Anantaram Balakrishnan and Christian V. Karsten. Optimal selection of liner containership services with limited transshipments. *European Journal of Operations Research*, 263:652–663, 2017. doi: https://doi.org/10.1016/j.ejor.2017.05.031.

Berit D Brouer, J Fernando Álvarez, Christian EM Plum, David Pisinger, and Mikkel M Sigurd. A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2):281–312, 2014a.

Berit Dangaard Brouer, Guy Desaulniers, and David Pisinger. A matheuristic for the liner shipping network design problem. *Transportation Research Part E: Logistics and Transportation Review*, 72:42–59, 2014b.

Berit Dangaard Brouer, Guy Desaulniers, Christian Vad Karsten, and David Pisinger. A matheuristic for the liner shipping network design problem with transit time restrictions. In *Computational Logistics*, pages 195–208. Springer, 2015.

Berit Dangaard Brouer, Christian Vad Karsten, and David Pisinger. Big data optimization in maritime logistics. In Ali Emrouznejad, editor, *Big Data Optimization: Recent Developments and Challenges*, volume 18 of *Studies in Big Data*, pages 319–344. Springer International Publishing, 2016.

Berit Dangaard Brouer, Christian Vad Karsten, and David Pisinger. Optimization in liner shipping. *4OR*, page submitted, 2017.

Marielle Christiansen, Kjetil Fagerholt, and David Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228(3):467–483, 2013.

Pierre Hansen and Nenad Mladenovic. Variable neighborhood search. In E. K. Burke and G. Kendall, editors, *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, pages 313–337. Springer, New York, 2014.

Christian V. Karsten, David Pisinger, Stefan Ropke, and Berit D. Brouer. The time constrained multi-commodity network flow problem and its application to liner shipping network design. *Transportation Research Part E: Logistics and Transportation Review*, 76:122–138, 2015.

Christian Vad Karsten, David Pisinger, Berit Dangaard Brouer, and Guy Desaulniers. Time constrained liner shipping network design. Transportation Research Part E: Coordination and Control in Transport Logistics, 2016.

Chung-Yee Lee and Dong-Ping Song. Ocean container transport in global supply chains: Overview and research opportunities. *Transportation Research Part B: Methodological*, 95:442–474, 2017.

Qiang Meng and Shuaian Wang. Optimal operating strategy for a long-haul liner service route. *European Journal of Operational Research*, 215(1):105–114, 2011.

Qiang Meng, Shuaian Wang, Henrik Andersson, and Kristian Thun. Containership routing and scheduling in liner shipping: overview and future research directions. *Transportation Science*, 48:265–280, 2014.

Benoit Montreuil. Towards a physical internet: meeting the global logisitcs sustainability grand challenge. *Logistics Research*, 3:71–87, 2011.

David Pisinger. Liner shipping network design – a new decomposition. In *Conference Handbook EURO-2016, 3–7 July, Poznan, Poland*, 2016.

Christian E.M. Plum. Network design in Maersk Line. 2015.

Christian EM Plum, David Pisinger, and Mikkel M Sigurd. A service flow model for the liner shipping network design problem. *European Journal of Operational Research*, 235(2):378–386, 2014.

Line Blander Reinhardt and David Pisinger. A branch and cut algorithm for the container shipping network design problem. *Flexible Services and Manufacturing Journal*, 24(3):349–374, 2012.

David Ronen. Cargo ships routing and scheduling: Survey of models and problems. *European Journal of Operational Research*, 12(2):119–126, 1983.

David Ronen. Ship scheduling: The last decade. *European Journal of Operational Research*, 71(3):325–333, 1993.

Zhuo Sun and Jianfeng Zheng. Finding potential hub locations for liner shipping. *Transportation Research Part B: Methodological*, 93:750–761, 2016.

Jesper Thorsen and Alexander Krogsgaard. Route network design for liner shipping. Master's thesis, DTU Management Engineering, 2016. Produktionstorvet 424, DK-2800 Kgs. Lyngby.

Unctad. Review of maritime transport. Technical report, `unctad.org/en/PublicationsLibrary/rmt2015_en.pdf` [Accessed: 30 June 2016], 2015.

Unctad. Review of maritime transport. Technical report, 2016. `unctad.org/en/PublicationsLibrary/rmt2016_en.pdf` [Accessed: 30 June 2016].